VLSI Memo No. 89-552
June 1989

Retrieving and Integrating IC Fabrication Data from Dissimilar Databases

Michael P. Ruf

Abstract

Factory personnel need to access data from many aspects of the fabrication environment. Many IC fabrication facilities store manufacturing data in distributed, heterogeneous database networks. Retrieval and integration of data can be a cumbersome task due to this configuration. The ideal solution to these problems is to standardize a data model, storing the manufacturing data in a single database. However, since such a standardization is not likely to occur in the near future, a more immediate solution is needed. This article discusses a system addressing these problems: DRIFS - *A Data Retrieval Interface for Integrated Circuit Fabrication Systems*. A uniform query interface and data model is defined for heterogeneous, distributed fabrication databases. A DRIFS prototype is described and evaluated.

Microsystems Research Center
Room 39-321
Massachusetts Institute of Technology
Cambridge Massachusetts 02139
Telephone (617) 253-8138

Acknowledgements

This research was supported in part by the Defense Advanced Research Projects Agency under contract number MDA972-88-K-0008.

Author Information

Ruf, *current address*: Rational, 3320 Scott Blvd., Santa Clara, CA 95054-3197.
(408) 496-3600.

Retrieving and Integrating IC Fabrication Data from Dissimilar Databases

by
Michael P. Ruf

Massachusetts Institute of Technology
Cambridge, MA 02139

June 7, 1989

Abstract

Factory personnel need to access data from many aspects of the fabrication environment. Many IC fabrication facilities store manufacturing data in distributed, heterogeneous database networks. Retrieval and integration of data can be a cumbersome task due to this configuration. The ideal solution to these problems is to standardize a data model, storing the manufacturing data in a single database. However, since such a standardization is not likely to occur in the near future, a more immediate solution is needed. This article discusses a system addressing these problems: DRIFS - *A Data Retrieval Interface for Integrated Circuit Fabrication Systems*. A uniform query interface and data model is defined for heterogeneous, distributed fabrication databases. A DRIFS prototype is described and evaluated.

1 Introduction

With the growing complexity of integrated circuit processes, the increasing number of technologies and product lines, and the relentless demand for higher volumes, commercial IC fabrication is increasingly dependent on information management. Information is collected from almost every aspect of the manufacturing process. Financial, marketing, design, engineering, production, scheduling, resource management, parametric, and control information all contribute to a vast collection of fabrication data. To maintain high productivity, IC manufacturers must draw on technical, historical, and corporate data at all levels of the fabrication process. More importantly, this data must be integrated and easily retrievable.

The past decade has seen the growth of commercial MIS systems for fabrication data. These systems are responsible for handling information for all aspects of the fabrication cycle, from design through packaging. However, in most cases, the systems are piecemeal, covering only a particular stage or aspect of the fabrication process. For example, separate computers and software are commonly used for logic design, process design, scheduling, fabrication, and facility support. Because these systems do not share data or communicate efficiently with each other, they do not support integration of data at a global level. Furthermore these large conglomerations of data are not always equipped with efficient retrieval interfaces. Due to the magnitude and complexity of such systems, often only

highly trained individuals can access and interpret the data. Improving the inadequacies of existing IC fabrication management systems will be a key factor in adapting to the growing information needs of factories of the future.

2 Fabrication Data in Heterogeneous Databases

The majority of the fabrication data is usually managed by a shop floor control (SFC) system. Several shop floor control (SFC) systems for IC fabrication are in use today. PROMIS[1] and COMETS[2], the two leading commercial manufacturing systems use table-based databases. SFCs normally handle a major portion of the information needs for a particular facility, but concentrate on data directly associated with the operation of the IC plant. This data can be separated into three major categories: facilities, engineering, and lot tracking/history.

Supplemental systems are used to collect and manage additional information not covered by shop floor control systems. Such systems might specialize in the following areas: testing/yield, scheduling, equipment maintenance, equipment control, financial/marketing, and facility support. The shop floor databases and the databases which store additional manufacturing data will be referred to as *local databases*. Often, users want to perform queries which draw on data from two or more of these local databases. How these queries are accomplished is an integration issue. One common method of integrating the data is to download information from supplemental databases into the main shop floor control database. This method has a number of disadvantages:

- Any outside information integrated into the SFC system must conform to the structure of the SFC database. If this structure cannot accommodate a certain data model, part of the data may be forfeited in converting to a suitable model.
- Integration is not real-time. An intermediate program must port the data from supplemental database to an SFC database. Queries may generate outdated information because the intermediate program can only be run periodically.
- As applications change, or new applications are written, integration needs may change. To restructure the integration model, the schema of the SFC databases must be modified. This is a high overhead procedure. Therefore, this integration method is not dynamic with regard to information needs.
- Because all access to integrated information must be through the SFC system, the demands on the system increase as more information from other databases is duplicated in the SFC databases to support integration.

[1] PROMIS is a registered trademark of PROMIS Systems Corporation.
[2] COMETS is a registered trademark of Consilium, Inc.

Because different people associated with the fabrication process may want access to different types of information, it is helpful to categorize these people into several groups with similar information needs. A typical breakdown is: equipment operators, engineers, manufacturing managers and supervisors, quality control personnel, production planners, support and maintenance personnel, and top level managers. Most shop floor control systems provide reports tailored for certain groups listed above, but cannot provide each group with a separate "view" of the underlying schema. Hence, new reports are difficult to generate, since retrieval interfaces are not designed to address the specialized interests of each group.

3 Multi-database Techniques

There are two prevalent approaches to integrating MDBSs. One approach is to provide a global schema which defines an integrated view of all data[1],[2],[3],[4],[5],[6],[7]. Under this configuration, the details of the local databases are hidden from the global user, who conceptually manipulates a classical homogeneous database. The second approach is to require explicit manipulation of the local databases, but provide mechanisms for sharing information between them[8],[9],[10].

An advantage of the global schema approach is that the LDBs can retain complete autonomy. Furthermore, retrieve-only integration of existing databases can be achieved without modifying or extending their schemata or functionality. Integrated update under the global schema configuration is a more complex issue, since update instructions must be applied to the global schema, presenting open problems associated with concurrency and redundancy. Several retrieve-only interfaces to MDBSs have been designed to generally address issues of the global schema approach. Among these are Multibase[1], Sirius-Delta[4], Mermaid[5], R*[6], and Gestalt[7].

Multibase, developed by the Computer Corporation of America, is centered around the ADAPLEX[3] database language, also developed by CCA. Multibase is designed to accommodate a broad range of data models in the underlying local databases. To accomplish this, it uses its own DBMS (the "Local Data Manager") to manipulate and process data from the LDBs. Sirius-Delta, a prototype system developed at INRIA, aims to allow users to manipulate existing distributed data as a unique database. It is implemented using the existing services of the LDBs. Mermaid, developed at Unisys Corporation, is intended for relational database management systems and uses existing DBMSs to manipulate and process the LDB data. R* is a prototype designed at the IBM Almaden Research Center to support transparent access to distributed relational databases through an extension of SQL. Gestalt is the central information-support system for the CAF Project (Computer-Aided Fabrication of Integrated Circuits) at M.I.T. It is designed to support read/write access to a broad range of underlying data structures, from conventional objects like personnel and machine records to unconventional objects like wafer models, IC masks, process flow programs, etc.

The integration of MDBSs can be accomplished in two stages[1],[4],[5]. First, a uniform data model and query interface are provided for retrieving data from each of the local databases. Second, an integrated representation is defined to allow access to multiple local databases via a single global query. This two-stage approach generally requires at least one new schema to be defined using the uniform data model for each of the existing LDBs. These new schemata provide homogeneity, since each of the heterogeneous LDBs are represented at this level using the same data model. Once the LDB schemata have been expressed in the uniform data model, a global schema can be built to integrate the data. The user may then submit queries in terms of the global schema, which conceptually represents an integrated homogeneous database. Executing these global queries involves solving several problems, including: query decomposition and optimization[5], query translation from the uniform data model to the individual data models of the LDBs[2], resolving semantic conflicts[1], providing sufficient concurrency control, and adequate performance.

4 A Testbed for Integrating Heterogeneous Fabrication Databases

The Texas Instruments Dallas Logic II (DLOG-II) IC fabrication facility was used as a testbed for a prototype implementation of DRIFS. At the time of the research DLOG-II was a low volume, fast turn-around manufacturing facility. A significant amount of manufacturing was dedicated to special work requests such as process development experiments. Because of the developmental nature of lot fabrication at DLOG-II, the make-up of the fabrication data emphasized engineering and parametric data rather than financial and planning data. The main computing resource at DLOG-II was a VAX 8650 running at 6 MIPS with 128 MB of memory, managing the following databases:

- **PROMIS.** The shop floor control database. By far the largest and most involved system, demanding a major portion of the computing time.
- **Engineering Test Database.** Stores test results from various integrated circuit test equipment.
- **Engineering Data Collection Database.** Stores supplemental engineering information used for generating charts and trend analysis.

Additionally, DLOG-II maintained several supplemental databases on mainframe, mini- and microcomputers. These include a facility support/control system, equipment control and particle monitoring databases. Representations of the DLOG-II PROMIS Database and the Engineering Test Database were defined in a DRIFS prototype. Since DLOG-II does not maintain a financial database, a simple mock one was created with Ingres, a relational database management system. The financial database was created on a VAX 785 running

ULTRIX[3], an implementation of UNIX[4] commonly used on DEC equipment. A separate computer and operating system were chosen for the financial database to demonstrate how DRIFS would perform in a distributed hardware and heterogeneous operating system environment.

4.1 Structure and Content of PROMIS Database

The PROMIS database consists of about 30 internally managed ISAM[5] files[11]. Most of the data they manage can be divided into three categories: facility information, engineering, and lot tracking and history.

The most extensive and significant facility information is stored in four hierarchically related tables describing production areas, work centers, equipment types and equipment units. Production areas are groups of work centers. Work centers and equipment types are groups of equipment units.

Most of the engineering data in PROMIS is associated with IC process specification. PROMIS records processing instructions defined by engineers and relays it to operators or automated fabrication equipment. PROMIS breaks down process specification into four levels of detail: device, process, recipe and operation. The device is at the top of the hierarchy, providing the most general information. Each of the next levels incorporates a greater amount of detail regarding manufacturing specifications for the device. The operation is at the bottom of the hierarchy, giving the highest level of detail. Once all the necessary modules have been defined, they are combined to from a complete manufacturing process flow.

Device records most commonly describe summary level instructions for fabricating manufacture IC's. The information in a device record can be divided into 3 major categories: administrative, parametric, and instructional. Administrative data contains descriptive information such as dates, the device record's history, personnel associated with the device, categories, etc. Parametric data consists of parameter names and values. Parameters may specify such things as lithography masks, probe tests, equipment settings, etc. They allow a general description of a device type to be tailored for specific devices. Instructional data describes the manufacturing flow for the device. Instructions can specify how to start making a device, specify a process flow, describe inventories for the device, or describe a "nested" device. All devices have an instruction of the first type, and most have a final inventory instruction. Some devices, such as purchased parts, do not have process instructions. Table 1 describes some of the fields composing the DEVC record.

PROMIS process records provide a higher level of detail than the devices which reference them. Several different devices may reference the same process. In this situation,

[3] ULTRIX is a registered trademark of Digital Equipment Corporation.

[4] UNIX is a registered trademark of AT&T.

[5] Indexed sequential access method.

Field Name	Type	Description
ACTIVFLG	Char.	Flag indicating whether this is the active version of the device.
DEVFUNCNAME	Char.	Name of the Device.
FROZEN	Byte	Flag indicating whether device can be modified.
PRODSTATUS	Char.	Status code indicating availability of device for production.
CREATEDT	Date	Date and time when this device record was created.
ACTIVEDT	Date	Date and time when this device record was made active
CHANGEDT	Date	Date and time of last modification to this device.
DESCR	Char.	Textual description of device.
INSTCOUNT	Integer	Number of instructions in the device record.
INSTTYPE	Char. Array	Instruction type (starting material, process, inventory, etc.)
INSTCOMMENT	Char. Array	Comment for instruction.
INSTPROID	Char. Array	Process ID used when instruction type is PROCESS.
INSTINVENTORYID	Char. Array	Inventory ID used when instruction type is INVENTORY.
NENGPARMS	Integer	Number of engineering parameters for this device.
NPLANPARMS	Integer	Number of planning parameters for this device
PARMNAME	Char. Array	Device parameter name.
PARMVAL	Char. Array	Device parameter value.

Table 1: Description of Selected Fields in the PROMIS Device Record.

each device applies its own set of devices parameters to the process. Different sets of processes are defined for each type of IC (e.g. *CMOS*, *NMOS*, *PMOS*, *BIPOLAR*). Processes typically have between 50 and 100 steps, each one naming a recipe to be used at that point. A recipe outlines a sequence of operations on a lot at a particular work center on a particular equipment type. Each step, or operation of a recipe describes a set of actions to be performed by equipment operators. Operations are the most detailed building blocks for processing specifications. They contain parametric and textual instructions for operating fabrication equipment.

PROMIS continually collects information on each lot as it is processed. This information, called lot data, is stored in two files: the active lot file and the lot history file. The active lot file contains one record for each lot, describing what is currently happening to the lot. Active lot records are updated at each work center. The lot history file stores an account of what happened to a lot at each step of processing. Each time a lot is tracked into a new work center, lot history entries are recorded.

4.2 Structure and Content of Engineering Test Database

The engineering test database (TDB) was designed and implemented by DLOG-II to manage control test data from IC test equipment. Each kind of tester at DLOG-II produces a different type and format of data. In fact, data formats can change from session to session on the same tester. The engineering test database consolidates the floating format test data from all the machines, and provides a standard query mechanism to the data.

The TDB is stored in VAX datalog format. Datalog files are composed of sequential ASCII variable length text records. There is one datalog file for each test session performed on each lot by each tester. The ASCII datalog files contain three types of records: data, header and format. There are two types of data records: test session data records, and test value data records. Test session data records contain the information necessary to locate a set of test data for any given wafer. Test value data records contain the test results for a given test session. A single test session data record precedes each set of test value data records. Header records define a format for the test session records. Format records define the format for test value data records. They specify field names which describe the values in the data records that follow. Figure 1 is an example of a datalog TDB file with header, format, test session data, and test value data records.

```
HDR1 TESTER TECH DEVICE LOT WAFER TEMP STATUS TEST-DATE TEST-TIME
FMT2 SUPPLY
FMT3 ICC2 ICC3 ICC5 VIL VIH VOH VOL
DAT1 SENTRY50 DMOS DPU-1 137380 23 25C PREBURN_IN 861231 14:49:34
DAT2 LHH
DAT3 832.0E-03 -2.000E-03 174.14E-03 1.399 1.99 1.988 338.1E-03
DAT3 821.0E-03 -2.000E-03 175.14E-03 1.320 1.98 1.981 340.2E-03
:
```

Figure 1: Example of a TDB file.

4.3 Structure and Content of Financial Database

Because DLOG-II does not maintain a networked financial database, a small database was created using one Ingres relation. The relation stores fictional flow cost and yield data associated with each device. The relation used to store the data is composed of the following domains:

- **devid.** Device ID.
- **nobars.** Number of bars, or die, per slice.
- **cqflowcost.** Average flow cost per slice for the current quarter.
- **cqyld.** Average yield for the current quarter.
- **pqflowcost.** Average flow cost per slice for the previous quarter.
- **pqyld.** Average yield for the previous quarter.

The the values for the devid field were chosen to match the DEVNAME field[6] from selected devices in the PROMIS database. The values for the remaining fields were supplied by the modified cost data.

[6] See Table 1.

4.4 Retrieval Mechanism for PROMIS

A PROMIS module, DATALINK, allows extraction and manipulation of data from any PROMIS file. Once selected, the DATALINK menu provides general extraction functions as well as data conversion functions. The General extract function provides an interactive method of retrieving data from individual PROMIS files. The extracted data is stored in an ascii work file in the user's PROMIS directory.

After initiating the General extract function, the user must specify extraction and search criteria. The extraction criteria identify the fields in a particular file to be extracted, while the search criteria, or query constraints, specify which entries to extract. Search criteria are entered as a field name, a relational operator, and a value. Once the extraction is complete, PROMIS can convert the work file to several different formats, including the standard data interchange format (DIF).

4.5 Retrieval Mechanism for the Engineering Test Database

The engineering test database provides a program, TDBEXTR, to extract data. The extract criteria, or query constraints, can be entered from the VMS command line. TDBEXTR generates three output files: a description file, a data file, and a log file. The extract criteria are specified using a simple extract language. Each criterion begins with a field name and is followed by a list of values for that field, enclosed in parenthesis. The values can be singular, a list separated by commas, a range separated by two periods, or a wild card expression.

4.6 Retrieval Mechanism for the Financial Database

Data is retrieved from the financial database using QUEL[12], the standard query language for Ingres. Query constraints in QUEL are called clauses. Each clause consists of a pair of expressions separated by a comparison operator. Basic queries can be formulated using only constant and attribute expressions. Attributes take the form *variable.domain*, where *variable* specifies a particular relation, and *domain* identifies a field in that relation. A typical clause might consist of an attribute followed by a comparison operator followed by a constant. For example, $cost.nobars < 200$ identifies the Cost Relations in which the field *nobars* contains a value less than 200. QUEL clauses can be linked together with logical operators (and, or, not) to form a qualification. An Ingres retrieve command specifies the relation and domains to extract from each tuple, as well as a qualification indicating which tuples to retrieve. Normally, Ingres will print the results of the query on the screen, but an optional argument to the retrieve command can specify a new relation to hold the output data. Ingres also provides a copy command to port data from an ingres relation to an outside file.

5 Overview of DRIFS

DRIFS[13] is a retrieve-only interface to heterogeneous, distributed fabrication databases. It is designed using the global schema approach to provide an integrated data representation without requiring changes to the existing local databases. It provides a software environment on a separate computer system, defining and storing a representation of each fabrication database. Using those representations, it generates queries particular to each fabrication database and retrieves data from them via computer networks.

5.1 DRIFS Schema Levels

In order to separate the tasks of providing homogeneity and integration, DRIFS breaks data acquisition into three levels: the local database level, the primitive level, and the user level. The local database level consists of the shop floor control system database and the supplemental databases.

For each local database schema, the primitive level stores a translation to the DRIFS data model. In answering queries posed at the primitive level, data is transferred from the underlying local database level through various networking techniques. The primitive level provides homogeneity, because it essentially maps each data model of the heterogeneous local databases to the uniform DRIFS data model. This is an involved task, since many issues must be considered, such as computer networking, data models and query languages at the local level, and data formats of output files at the local level. While data representations from separate local databases are all stored using the DRIFS data model, they are not integrated at this level.

Once the data has been mapped to the uniform data model of the primitive level, it can be integrated at the user level. This level combines representations from the primitive level to allow data from various local databases to be integrated in common structures. User level data representations can be tailored for specific groups such as planners or engineers, allowing each group to have separate "views" into the primitive level.

5.2 DRIFS Data Structures

The DRIFS data model is expressed as *primitive structures*, which are templates for data retrieved from the local databases. Each structure consists of several titled slots where atomic data is stored. Each slot is defined with the following fields: slot name, slot type, list singleton identifier. The slot name describes what kind of data is stored in the slot, the slot type specifies what format the data is stored in (i.e. string, number), and the list/singleton identifier indicates whether the data is stored as a list of values or as a single value. Associated with each primitive structure is information specifying from which local database to retrieve its data and how to query that database. This information indicates in which relation, file, record, etc. the data is stored. It also defines a variable map linking

each slot in the primitive structure to a particular field in the local database. Primitive structures are grouped by local database, and the collection of all primitive structures for a particular local database makes up DRIFS's representation for that database.

The user level provides the means for integrating related data from each local database. User level structures combine data from the primitive structures by incorporating relevant slots from each primitive structure. The primitive structures which comprise a particular user level structure must have at least one slot in common. These key slots are used to combine the results of primitive level queries.

6 The DRIFS Prototype Implementation

DRIFS was prototyped on a Texas Instruments Explorer LISP machine. The Explorer is a single-user workstation designed for rapid development and prototyping in a symbolic processing environment. Running Common LISP with incremental garbage collection, the Explorer can manage up to 128 megabytes of virtual memory. The internal Nubus architecture uses a 32-bit LISP processor running at 10 megahertz (100 ns clock period). An ethernet controller is provided for communications with local area networks. The Explorer provides several networking services, including transparent file I/O, remote login, and task-to-task communication. To support these services, the Explorer uses the multiple communication protocols. Figure 2 shows the networking configuration for the Explorer used to prototype DRIFS. The networking names of the machines are shown in parenthesis.

6.1 DRIFS Software Environment

The DRIFS prototype was implemented using windows and pop-up menus in which primitive and user level structures are created and manipulated. To define a structure, the user specifies a DRIFS structure name, a local database, a query type for that database, and each slot in the structure. DRIFS then creates a LISP flavor as a template for data retrieved from the local databases. When data is retrieved from a local database it is parsed into primitive structures, and stored in flavor instances for later viewing.

DRIFS identifies a set of query types associated with each local database: PROMIS, engineering test, and financial. Ideally, only one query type would be needed for each database. However, since the file structure for the PROMIS database is not consistent, a separate query type was defined for reading files with multiple record types. Slots are defined by specifying a slot name, slot type, and list/singleton specifier. The slot name is used only by DRIFS as a reference and the slot type indicates the type of data which will be stored in the slot.

Once the primitive structure has been defined, the user must create a mapping which associates each slot in the structure with a specific field in the PROMIS database file. The PROMIS fields are selected using pop-up menus. Since PROMIS treats array fields

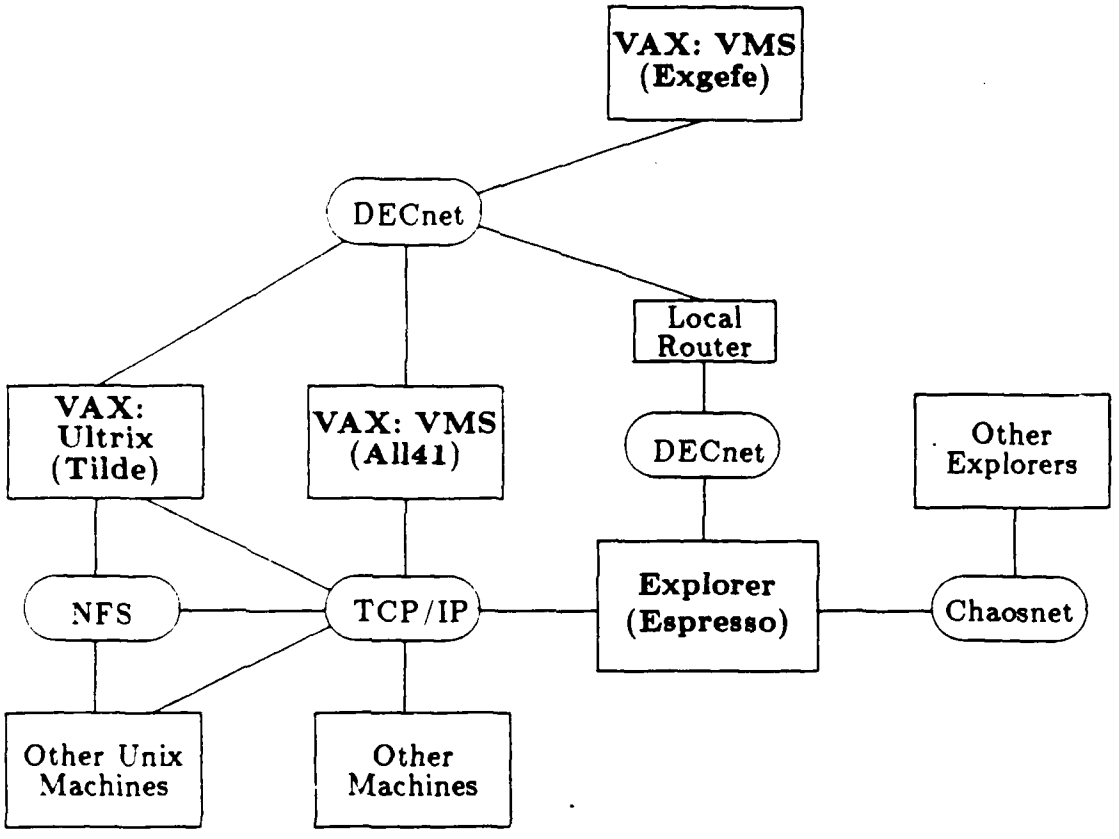

Figure 2: Partial schematic representation of Ethernet network at TI Computer Science Center

The networking name of each computer is shown in parenthesis.

differently from other fields, they must be handled specially by DRIFS. When an array field is selected in a mapping, DRIFS asks for the maximum number of elements to retrieve from the array and for the PROMIS field which specifies exactly how many elements are in the array.

User level structures combine slots from primitive structures, allowing data from separate databases to be integrated in a common data model. They are created using pop up menus that specify the component primitive structures and which slots to include from each one. One slot from each primitive structure must be identified as a key slot. DRIFS expects the data in key slots to be stored in the same format across primitive structures.

6.2 Retrieving Primitive Level Data

To initiate a DRIFS primitive level query, the user selects a primitive structure to retrieve and specifies the search criteria (known to DRIFS as query constraints) via pop-up menus. Once the query constraints have been specified by the user, DRIFS translates them into the query language of the local database and issues them to the server for that database. Two methods were evaluated for communicating query commands to local database servers: remote batch jobs and ascii-translating character streams.

Batch jobs have the advantage of standard handshaking provided by the network protocols, allowing for better handling of error conditions, and less programming overhead. However, response time for batch jobs can be unacceptably slow, depending on the configuration of the host computer. For example, EXGEFE (the host computer for PROMIS and the engineering test database at DLOG-II) gives batch jobs minimal priority at peak computing hours. In fact, during certain times of the day, batch processing virtually stops until the computing load is relieved. Another disadvantage to remote batch jobs is that they must initiate and release a new process each time a retrieval is requested. A large portion of the time it takes to run a PROMIS query is consumed in getting to the proper PROMIS menu and exiting the system. With batch jobs, this must be done each time a query is run.

In contrast, an ascii-translating character stream is a direct connection to another computer via remote login. The stream has an output buffer through which a program can send commands to the remote host as if a user were typing them. The output from the host is piped into the stream's input buffer where it can be interpreted by DRIFS. Ascii-translating character streams have the advantage of efficiency but are more prone to unanticipated errors. For example, if the host broadcasts a system message to all users, DRIFS may not know how to interpret that message, or, if a process times-out or aborts for any reason, DRIFS may continue to send commands to the non-existent process, not knowing it has been deactivated. Also, if a stream has been inactive for a certain length of time, the remote login session may terminate automatically in what is called an autologout. Although procedures may be included to detect and recover from such conditions, it is difficult to anticipate all failure modes.

Ascii-translating streams were used to communicate with the PROMIS and the engineering test databases on EXGEFE, and batch jobs were used to communicate with the financial database on TILDE. Batch jobs were acceptable for TILDE because even at high system load, there was little or no time between queuing and execution. Since direct network connections for ascii-translating streams may not exist from the retrieval computer to the local host, DRIFS defines a connection path which lists the hosts, or connection nodes, leading to the desired host. The ascii-translating character stream must login to each node separately until it reaches the desired host. The connection nodes hold the information necessary for login at each host.

6.3 Retrieving Primitive Structure Data

Note from Figure 2 that there is more than one network path between ESPRESSO and EXGEFE. To generate a retrieval connection, ESPRESSO can use DECnet expressly to connect to the local router, and then to EXGEFE, or it can use TCP/IP to connect to ALL41 and then login to EXGEFE through DECnet. Because the Explorer provides more comprehensive and reliable support for TCP/IP streams, connection nodes were defined to use the latter path.

To minimize use of the character stream, DRIFS writes its queries to a script file on EXGEFE and uses the stream only to activate and monitor the progress of the script command. Due to a problem with the Explorer implementation of DECnet, files longer than about 600 bytes could not be transferred directly from ESPRESSO to EXGEFE. To work around this problem, script files were transferred indirectly to EXGEFE via ALL41. Each script file includes commands to perform the specified query and to convert the PROMIS output file into DIF format. Using DECnet, the DIF file is then read from EXGEFE and parsed into primitive structures.[7]

DRIFS uses a separate character stream connected to EXGEFE to query the engineering test database. This character stream communicates at the VMS command level, passing query constraints to the extract program (TDBEXTR) from the command line. Using DECnet, the description file is then read to identify the format of the data file. Then, the data file is read and parsed into primitive structures.

To query data from the financial database, DRIFS creates on TILDE a script file containing Ingres commands. A batch file is used to load Ingres and activate the script, which retrieves the requested fields and copies them to an output file. DRIFS reads the output file from TILDE, parsing the data into primitive structures.

[7] The Explorer DECnet failed only when transferring files *to* EXGEFE. Therefore, DECnet could be used to transfer the DIF file directly from EXGEFE to ESPRESSO.

6.4 The User Level Interface

6.4.1 Combining PROMIS and Financial Data

To describe the DRIFS user level interface, an example user level structure, FIN-DEVICE, will be used. It combines data from two primitive level structures: DEVICE from PROMIS (Figure 3) and DEVICE-COST-DATA from the financial database (Figure 4).

FIN-DEVICE (Figure 5) incorporates fields from it component primitive structures which might be of interest to a production planner. Thus, retrieving FIN-DEVICE structures requires data to be integrated from PROMIS and the financial database.

DRIFS divides a user level query into separate primitive level queries and uses the key slots to match corresponding results from each query. For example, to view all FIN-DEVICE's with device ID's beginning in "ASAM", DRIFS would perform two separate primitive level queries. First it would retrieve device data from PROMIS by querying all DEVICE structures with NAME's beginning in "ASAM". Then it would retrieve cost data from the financial database by querying all DEVICE-COST-DATA structures with NAME's beginning in "ASAM". At this point, DRIFS compares the data in the key slots (in this case, the NAME's) to match each DEVICE structure with its corresponding DEVICE-COST-DATA structure. Once these primitive structures are paired up, each pair is combined to form a FIN-DEVICE structure.

In the preceding example, the query constraints concerned the key slots of the DEVICE and DEVICE-COST-DATA structures. If no key slot was included in the constraints, the user level query would be handled differently. For example, to retrieve all FIN-DEVICE structures with current quarter yields less than 40%, DRIFS would first retrieve all DEVICE-COST-DATA structures from the financial database with current quarter yields less than 40%. Then, it would use the data in the DEVICE-COST-DATA key slots (in this case, the NAME's) to build the query constraints for the DEVICE structure. DRIFS would use those query constraints to retrieve from PROMIS a matching DEVICE structure for each DEVICE-COST-DATA structure it has already retrieved from the financial database. The pairs are then combined to form FIN-DEVICE structures.

6.4.2 Combining PROMIS and Engineering Test Data

Integrating data from PROMIS and the engineering test database cannot be handled the manner described in Section 6.4.1, because there is not a one-to-one correspondence between PROMIS entries and entries in the test database: For each lot in PROMIS, there are several entries in the entries in the test database; for example, suppose the user wants to combine data in the ACTIVE-LOT primitive structure from PROMIS (Figure 6) and the LOT-TEST-RESULT primitive structure from the engineering test database (Figure 7). There are many LOT-TEST-RESULT's which correspond to each ACTIVE-LOT. To handle this situation, DRIFS allows user level structures to identify *sub-structures*. Sub-structures are primitive structures, such as LOT-TEST-RESULT, which have a many-to-one correspondence to other primitive structures, such as ACTIVE-LOT. The user level

```
DEVICE (PROMIS)
                NAME:  string              SINGLETON
         DESCRIPTION:  string              SINGLETON
              ACTIVE:  string              SINGLETON
              FROZEN:  number              SINGLETON
              STATUS:  string              SINGLETON
         CREATE-DATE:  string              SINGLETON
         ACTIVE-DATE:  string              SINGLETON
       LAST-MOD-DATE:  string              SINGLETON
       INSTRUCT-TYPE:  string              LIST
    INSTRUCT-COMMENT:  string              LIST
    INSTRUCT-PROC-ID:  string              LIST
  INSTRUCT-INVENT-ID:  string              LIST
       NO-ENG-PARAMS:  number              SINGLETON
  NO-PLANNING-PARAMS:  number              SINGLETON
          PARAM-NAME:  string              LIST
         PARAM-VALUE:  string              LIST
```

Figure 3: Description of DEVICE primitive level structure.

```
DEVICE-COST-DATA (FINANCIAL-DB)
                NAME:  string              SINGLETON
       BARS-PER-SLICE:  number             SINGLETON
    CUR-QTR-FLOW-COST:  number             SINGLETON
         CUR-QTR-YIELD:  number             SINGLETON
   PREV-QTR-FLOW-COST:  number             SINGLETON
       PREV-QTR-YIELD:  number             SINGLETON
```

Figure 4: Description of DEVICE-COST-DATA primitive level structure.

```
FIN-DEVICE

Slots:
                NAME  (from DEVICE)
         DESCRIPTION  (from DEVICE)
              ACTIVE  (from DEVICE)
              FROZEN  (from DEVICE)
              STATUS  (from DEVICE)
         CREATE-DATE  (from DEVICE)
         ACTIVE-DATE  (from DEVICE)
       LAST-MOD-DATE  (from DEVICE)
      BARS-PER-SLICE  (from DEVICE-COST-DATA)
   CUR-QTR-FLOW-COST  (from DEVICE-COST-DATA)
       CUR-QTR-YIELD  (from DEVICE-COST-DATA)
  PREV-QTR-FLOW-COST  (from DEVICE-COST-DATA)
      PREV-QTR-YIELD  (from DEVICE-COST-DATA)

Key Slots:
                NAME  (from DEVICE)
                NAME  (from DEVICE-COST-DATA)
```

Figure 5: Description of FIN-DEVICE user level structure.

structure, ENG-ACTIVE-LOT (Figure 8), incorporates slots from ACTIVE-LOT which are of interest to an engineer. It also incorporates lot test data associated with each active lot by identifying LOT-TEST-RESULT as a sub-structure.

7 Evaluations

7.1 Evaluation of DRIFS Prototype Implementation

The DRIFS prototype is successful in providing a standard data model and a simplistic retrieval interface to data in heterogeneous fabrication databases. Because the mock financial database resides on a separate computer system from the DLOG-II VAX which stores the PROMIS and engineering test databases, the prototype demonstrates that the DRIFS concept can be implemented in a heterogeneous hardware environment. The window-oriented menu interface of the DRIFS prototype provides a flexible and convenient method for defining representations of the local fabrication databases.

The model for DRIFS primitive structures is sufficient for defining representations of each of the local fabrication databases. In addition to providing homogeneity, the primitive structures are easily integrated at the user level. The use of key slots and substructures allows primitive data from separate fabrication databases to be integrated via user level structures. Additionally, user level structures provide a means of defining specialized

```
ACTIVE-LOT (PROMIS)
                NAME:  string              SINGLETON
             COMMENT:  string              SINGLETON
         DEVICE-NAME:  string              SINGLETON
    CUR-PROCESS-NAME:  string              SINGLETON
     CUR-RECIPE-NAME:  string              SINGLETON
  CUR-PROD-AREA-NAME:  string              SINGLETON
 CUR-WORKCENTER-NAME:  string              SINGLETON
 CUR-EQUIP-TYPE-NAME:  string              SINGLETON
 CUR-EQUIP-UNIT-NAME:  string              SINGLETON
   COMPLETION-CLASS:  string              SINGLETON
             STATE:  string              SINGLETON
   STATE-ENTRY-TIME:  string              SINGLETON
             STAGE:  string              SINGLETON
      TRACKING-STAGE:  string              SINGLETON
     CUR-STEP-NUMBER:  number              SINGLETON
     END-STEP-NUMBER:  number              SINGLETON
     EMPL-ID-TRACKIN:  string              SINGLETON
    EMPL-ID-TRACKOUT:  string              SINGLETON
        STEP-COMMENT:  string              SINGLETON
          QUEUE-TIME:  string              SINGLETON
     STEP-START-DATE:  string              SINGLETON
       STEP-END-DATE:  string              SINGLETON
       RELEASE-TIME:  string              SINGLETON
     STEP-START-SIZE:  number              SINGLETON
     MECH-STEP-YIELD:  number              SINGLETON
        NO-OF-DIE-IN:  number              SINGLETON
      DIE-STEP-YIELD:  number              SINGLETON
  CUR-MECH-LOT-YIELD:  number              SINGLETON
   CUR-EFF-DIE-YIELD:  number              SINGLETON
```

Figure 6: Description of ACTIVE-LOT primitive level structure.

```
LOT-TEST-RESULT (TEST-DB)
            TESTER:  string              SINGLETON
        TECHNOLOGY:  string              SINGLETON
            DEVICE:  string              SINGLETON
          LOT-NAME:  string              SINGLETON
         TEST-DATE:  string              SINGLETON
         TEST-TIME:  string              SINGLETON
        TEST-TYPES:  string              LIST
      TEST-RESULTS:  TEST-RESULT         LIST
```

Figure 7: Description of LOT-TEST-RESULT primitive level structure.

```
ENG-ACTIVE-LOT

Slots:
                    NAME  (from ACTIVE-LOT)
                 COMMENT  (from ACTIVE-LOT)
             DEVICE-NAME  (from ACTIVE-LOT)
        CUR-PROCESS-NAME  (from ACTIVE-LOT)
         CUR-RECIPE-NAME  (from ACTIVE-LOT)
      CUR-PROD-AREA-NAME  (from ACTIVE-LOT)
     CUR-WORKCENTER-NAME  (from ACTIVE-LOT)
     CUR-EQUIP-TYPE-NAME  (from ACTIVE-LOT)
     CUR-EQUIP-UNIT-NAME  (from ACTIVE-LOT)
                   STATE  (from ACTIVE-LOT)
                   STAGE  (from ACTIVE-LOT)
         CUR-STEP-NUMBER  (from ACTIVE-LOT)
            STEP-COMMENT  (from ACTIVE-LOT)

Key Slots:
                    NAME  (from ACTIVE-LOT)

Sub-structures:
     LOT-TEST-RESULT  Key Slot: LOT-NAME
```

Figure 8: Description of ENG-ACTIVE-LOT user level structure.

"views" of the fabrication data for certain groups (i.e. engineers, production planners, etc.) This was demonstrated in the prototype through the ENG-ACTIVE-LOT and FIN-DEVICE user level structures, tailored for engineers and production planners, respectively.

While the Explorer LISP machine used to implement DRIFS provides an excellent environment for rapid prototyping, networking from the Explorer to other computers is slow and cumbersome. The work-around to the networking problem described in Section 6.3 adds substantial overhead to running PROMIS database queries from DRIFS. This overhead, combined with the already slow transfer rate (about 500 to 800 bytes/sec) between the Explorer and the DLOG-II VAX severely limits the performance of PROMIS queries from DRIFS. Table 2 shows the access times required to retrieve one DEVICE structure from PROMIS. Note that transferring the PROMIS script file to EXGEFE accounts for 68% of the total access time (due to the networking work-around). Since transfer time for script files is constant with respect to the number of DEVICE's retrieved from PROMIS, this percentage become less significant (15%) when a larger number (20) of DEVICE structures are retrieved (see Table 3).

	Transfer Script File to EXGEFE	Execute Script on EXGEFE	Read DIF Output File	Total Access Time
Trial 1	0:36	0:11	0:11	0:58
Trial 2	0:33	0:07	0:10	0:50
Trial 3	0:34	0:09	0:07	0:50
Trial 4	0:35	0:08	0:07	0:50
Average	**0:35**	**0:09**	**0:09**	**0:52**
Average %	**68%**	**17%**	**17%**	

Table 2: Access times (M:SS) to retrieve one DEVICE structure from PROMIS.

	Transfer Script File to EXGEFE	Execute Script on EXGEFE	Read DIF Output File	Total Access Time
Trial 1	0:38	1:46	1:55	4:19
Trial 2	0:41	2:19	2:00	5:00
Trial 3	0:40	1:46	1:59	4:25
Trial 4	0:39	1:49	1:58	4:26
Average	**0:40**	**1:55**	**1:58**	**4:33**
Average %	**15%**	**42%**	**43%**	

Table 3: Access times (M:SS) to retrieve 20 DEVICE structures from PROMIS.

Access times for queries to the engineering test and financial database are more acceptable. Table 4 shows the access times for retrieving test data for an active lot with 15

	Run TDB Extract on EXGEFE	Read TDB Output File	Total Access Time
Trial 1	0:19	0:07	0:26
Trial 2	0:17	0:10	0:27
Trial 3	0:20	0:08	0:28
Trial 4	0:22	0:08	0:30
Average	**0:20**	**0:08**	**0:28**
Average %	**71%**	**29%**	

Table 4: Access times (M:SS) to retrieve 15 LOT-TEST-RESULT structures from the TDB.

	Run TDB Extract on EXGEFE	Read TDB Output File	Total Access Time
Trial 1	0:31	0:25	0:56
Trial 2	0:27	0:27	0:54
Trial 3	0:23	0:27	0:50
Trial 4	0:22	0:33	0:55
Average	**0:26**	**0:28**	**0:54**
Average %	**48%**	**52%**	

Table 5: Access times (M:SS) to retrieve 114 LOT-TEST-RESULT structures from the TDB.

	Transfer Script File to TILDE	Execute Script on TILDE	Read Ingres Output File	Total Access Time
Trial 1	0:04	0:24	0:08	0:36
Trial 2	0:04	0:20	0:12	0:36
Trial 3	0:07	0:21	0:13	0:41
Trial 4	0:04	0:19	0:12	0:35
Average	**0:05**	**0:21**	**0:11**	**0:37**
Average %	**13%**	**57%**	**30%**	

Table 6: Access times (M:SS) to retrieve one DEVICE-COST-DATA structure from the financial database.

	Transfer Script File to TILDE	Execute Script on TILDE	Read Ingres Output File	Total Access Time
Trial 1	0:05	0:32	0:12	0:49
Trial 2	0:04	0:41	0:12	0:55
Trial 3	0:04	0:31	0:12	0:47
Trial 4	0:04	0:25	0:11	0:40
Average	**0:04**	**0:32**	**0:12**	**0:48**
Average %	**08%**	**67%**	**25%**	

Table 7: Access times (M:SS) to retrieve 10 DEVICE-COST-DATA structures from the financial database.

LOT-TEST-RESULT entries. Table 5 shows the access times for retrieving test data for an active lot with 114 LOT-TEST-RESULT entries. Notice that at 114 entries, reading the output file over the network is the dominant time factor. Table 6 shows the access times for retrieving the cost data associated with one device. Table 7 shows the access times for retrieving the cost data associated with 10 devices. The Ingres execution time increases from 21 sec. to 32 sec. (an increase of 52%) when the number of DEVICE-COST-DATA structures retrieved increases from one to 10 (an increase of 900%). These figures indicate that with small queries, a major portion of the access time is overhead in loading and executing the Ingres script file.

While DRIFS provides a standard retrieval interface and data model for heterogeneous fabrication databases, its effectiveness can be limited by the existing interfaces to the individual local databases and by networking demands. For example, the only interface to PROMIS data is through menus intended to handle interactive commands from a user console. Automated interaction with these menus is a cumbersome and error-prone method. DRIFS could be more effective if each local database provided a retrieval interface designed to handle automated data retrieval. Even without such interfaces, however, DRIFS can provide an excellent means for off line data retrieval in converting to a homogeneous fabrication database.

References

[1] J. Smith, P. Bernstin, U. Dayal, N. Goodman, T. Landers, K. Lin, and E. Wong, "Multibase–integrating hetergeneous distributed database systems," in *Proceedings of the National Computer Conference*, 1981.

[2] T. Landers and R. Rosenberg, "An overview of Multibase," in *Proceedings of International Symposium of Distributed Databases*, (Berlin, West C rmany), 1982.

[3] A. Chan, U. Dayal, and S. Fox, "An Ada-compatible distributed database management system," *Proceedings of the IEEE.*, vol. 75, pp. 674–694, May 1987.

[4] A. Ferrier and C. Stangret, "Heterogeneity in the distributed database management system sirius-delta," in *Proceedings Eighth International Conference on Very Large Data Bases*, (Mexico City), Sept. 1982.

[5] M. Templeton, D. Brill, S. Dao, E. Lund, P. Ward, A. L. P.Chen, and R. MacGregor, "Mermaid—a front-end to distributed heterogeneous databases," *Proceedings of the IEEE.*, vol. 75, pp. 695–708, May 1987.

[6] B. Lindsay, "A retrospective of R*: a distributed database management system," *Proceedings of the IEEE.*, vol. 75, pp. 668–673, May 1987.

[7] M. L. Heytens and R. S. Nikhil, "GESTALT: an expressive database programming system," December 1987. To be published.

[8] B. Czejdo, M. Rusinkiewiez, and D. Embley, "An approach to schema integration and query formulation in federated database systems," in *Proceedings 3rd International Conference on Data Engineering*, (Los Angeles), Feb. 1987.

[9] D. Heimbigner and D. McLeod, "A federated architecture for information management," *ACM Transactions on Office Information Systems*, vol. 3, pp. 253–278, July 1985.

[10] W. Litwin and A. Abdellatif, "An overview of the multi-database manipulation language MDSL," *Proceedings of the IEEE.*, vol. 75, pp. 621–632, May 1987.

[11] *PROMIS Standard System Guide.* PROMIS Systems Corporation, 4.2 ed., 1987.

[12] *INGRES Reference Manual.* Relational Technology, Inc., Berkeley, CA., 3.0, vax/vms ed., 1984.

[13] M. Ruf, DRIFS: — *A Data Retrieval Interface for Integrated Circuit Fabrication Systems.* Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, Jan. 1989.

Contents